Deepika Negi
CSP SPL-1
Roll No: 43.

PAGE No
DATE: / /202

Tutorial-2

Q1

1st time → i=1
2nd → i=1+2
3rd → i=1+2+3+1
ith time → i=(1+2+3+...+i)<n

$\Rightarrow \frac{i(i+1)}{2} < n$

$\Rightarrow i^2 < n$

$\Rightarrow i < \sqrt{n}$

Time complexity: $O(\sqrt{n})$ Ans

Q2 Recurrence Relation:

$f(n) = f(n-1) + f(n-2)$

Let T(n) denote the time complexity of f(n)
for f(n-1) & f(n-2) time = T(n-1) and T(n-2).
We have one more addition to sum and result
for n>1.

$T(n) = T(n-1) + T(n-2) + 1$ — (I)

for n=0 & n=1, no addition
∴ T(0) = T(1) = 0
Let $T(n-1) \approx T(n-2)$ — (II)
Put (II) in (I)

$T(n) = T(n-1) + T(n-1) + 1$
$= 2 \times T(n-1) + 1$

Using backward substitution
∴ $T(n-1) = 2 \times T(n-2) + 1$

$T(n) = 2 \times (2 \times T(n-2) + 1) + 1$
$= 4 \times T(n-2) + 3$

Now we can substitute
$T(n-2) = 2 \times T(n-3) + 1$
$T(n) = 8 \times T(n-3) + 1.$

General Equation :-
$T(n) = 2^k \times T(n-k) + (2^k - 1)$ — (iii)
for $T(0)$
$n - k = 0 \Rightarrow k = n.$

Substituting values in (iii)
$T(n) = 2^n \times T(0) + 2^n - 1$
$= 2^n + 2^n - 1$
$\boxed{T(n) = O(2^n)}$

Space complexity $= O(N)$

The function calls are executed sequentially. This execution guarantees that the stack size will exceed the depth of calls for 1st $f(n-1)$ it will create N stack frame, the other $f(n-2)$ will create N/2. so the longest is N.

Q3 i) $O(n \log n)$:-

```
#include <iostream>
using namespace std;
```

PAGE No.
DATE : / /202

```
int partition (int arr[], int start, int end)
{ int pivot = arr[start];
  int count = 0;

  for (int i=(start+1); i<=end; i++)
    { if (arr[i]<=pivot)
      count++;
    }
  int pivot_n = start + count;
  swap (arr[pivot_n], arr[start]);

  int i=start, j=end;
  while (i<pivot_n && j>pivot_n)
  { while (arr[i]<=pivot)
      { i++;
      }
    while (arr[j]>pivot)
      { j--;
      }
    if (i<pivot_n && j>pivot_n)
      { swap (arr[i++], arr[j--]);
      }
  }
  return pivot_n;
}
void quick (int arr[], int start, int end)
{ if (start >= end)
    return;
  int p= partition (arr, start, end);
  quicksort (arr, start, p-1);
  quicksort (arr, p+1, end);
}
```

```
int main ()
{ int arr [] = {6,8,5,2,1};
  int n = 5;
  quicksort (arr, 0, n-1);
  return 0;
}
```

(i) O(N3) -

```
int main ()
{ int n=10;

  for (int i=0; i<n; i++) {
  for (int j=0; j<n; j++) {
  for (int k=0; k<n; k++) {

    printf ("*");
  }
  }
  } return 0;
}
```

(iii) O(log (log n)) -

```
int count Prime (int n)
{ if (n<2)
    return 0;

  boolean [] non prime = new boolean [n];
  non prime [1] = true;
  int num non primes = 1;
  for (int i=2; i<n; i++)
```

PAGE No
DATE : / /202

```
if (nonprime[i])
continue;
int j = i*2;
while (j<n)
{ if (!nonprime[j])
{ nonprime[j] = true;
  numnonprime++;
}
  j+=i;
}
}
return (n-1)-numnonprime;
}
```

Q4 $T(n) = T(n/4) + T(n/2) + cn^2$

Using master's theorem.

We can assume $T(n/2) >= T(n/4)$

Equation can be re-written as

$T(n) <= 2T(n/2) + n^2$

$T(n) <= O(n^2)$

$T(n) \in O(n^2)$

Also $T(n) >= n^2 \Rightarrow T(n) = O(n^2)$

$\Rightarrow T(n) = \Omega(n^2)$

$\therefore T(n) = \Theta(n^2)$ & $T(n) = \Omega(n^2)$

$\boxed{T(n) = O(n^2)}$

Q5 for { i=1, n times (inner loop)
i=2, n/2 times
i=3, n/3 times

so, $n + \frac{n}{2} + \frac{n}{3} + \cdots + \frac{n}{n}$

$n\left(1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n}\right)$

$n \times \sum_{k=1}^{n} 1/k$

$n \times \log n$

Time complexity = $O(n \log n)$

Q6. for (int i=2); i<=n; i= pow(i,k)
{ // O(1)
};

iterations :

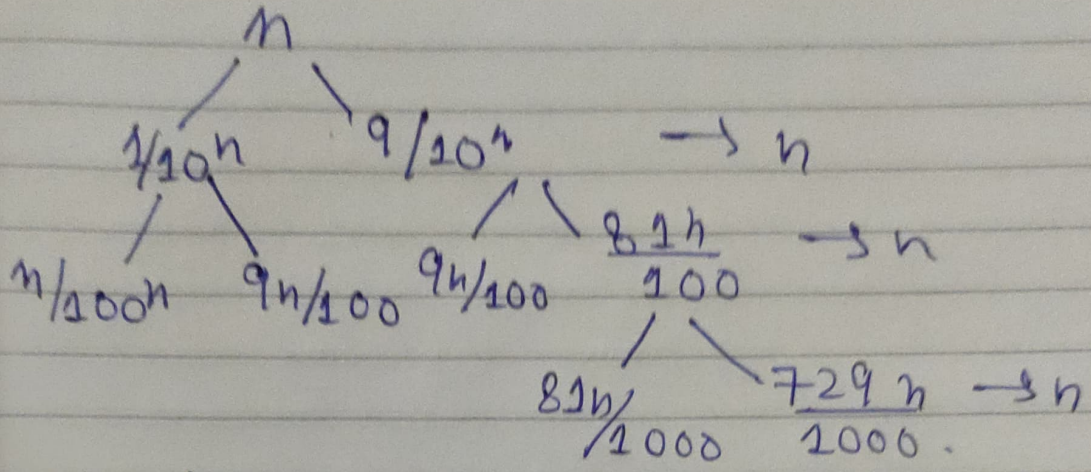
for 1st iteration → 2
2nd → 2^k
3rd → $(2^k)^k$
n → $2^{k \log_k (\log(n))}$.

∵ last term must be less than or equals to n

$2^{k \log_k (\log(n))} = 2^{\log n} = n$

Each iterations takes constant time
∴ Total iterations = $O(\log(\log(n)))$.

Q7



⇒ If we split this.
Recurrence relation → $T(n) = T\left(\frac{9n}{10}\right) + T\left(\frac{n}{10}\right) + O(n)$

where 1st branch is of size $\frac{9n}{10}$ and second is of $\frac{n}{10}$

Solving above using recursion tree approach :-
At 1st level, value = n
At 2nd level, value = $\frac{9n}{10} + \frac{n}{10} = n$.

Value remains same at all levels i.e. n.
Time complexity = summation of values.
$= O(n \times \log_{10/9} n)$ [upper bound]
$= \Omega(n \log_{10} n)$ [lower bound]
$\Rightarrow O(n \log n)$

Q6 a) $100 < \log(\log n) < \log n < (\log n)^2 < \sqrt{n} < n < n(\log n) < \log(n!) < n^2 < 2^n < 4^n < 2^{2n}$.

b) $1 < \log(\log n) < \sqrt{\log(n)} < \log n < \log 2n < 2(\log n) < n < n(\log_2 n) < 2n < \log(n!) < n^2 < n! < n \cdot 2^{2n}$.

c) $96 < \log_8 n < \log 2n < 5n < n(\log_6 n) < n(\log_2 n) < \log(n!) < 8n^{2.3} < 7n^3 < n! < 8^{\frac{2n}{2}}$